⚡ EV SOCIETY ⚡

EV BATTERY PACK SIMULATOR

Simulation • Backend API • Database • Interactive Dashboard

Prateek

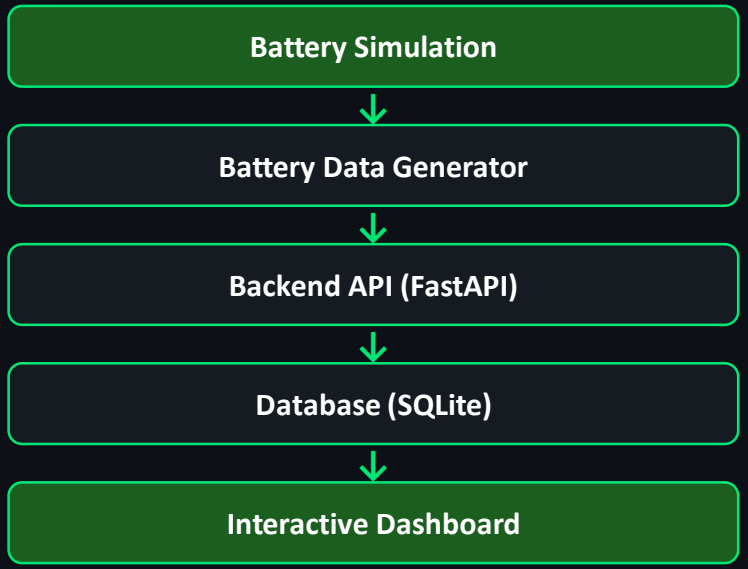
Internship Project

Python | FastAPI | SQLite | Streamlit

Project Overview

A software-based platform to generate, store, retrieve, and visualize battery simulation data — no physical hardware required.

- ⚡ Simulates electrical & thermal battery behavior in software
- Supports multiple battery chemistries (LFP, NMC811, NCA, LCO)
- REST API exposes battery data to any HTTP client
- Interactive dashboard for real-time analysis
- Persistent SQLite storage via SQLAlchemy ORM



Why Battery Simulation?

⚠ PROBLEM

- ✕ Expensive physical hardware
- ✕ Time-consuming test cycles
- ✕ Difficult to scale testing
- ✕ Hardware-dependent workflows
- ✕ Risky for edge-case experiments

✓ SOLUTION

- ✓ Generate any number of battery states
- ✓ Test application logic safely
- ✓ Experiment with chemistry configs
- ✓ Monitor parameters programmatically
- ✓ Visualize data instantly

Battery Parameters

SOC
State of Charge

Remaining charge level (0–100%)

SOH
State of Health

Health vs. original capacity

Voltage
Electrical Potential

Output voltage in Volts

Current
Charge Flow

Charge/discharge rate in Amps

Temperature
Thermal State

Cell temperature in °C

Capacity
Energy Storage

Maximum charge in Ah

Cycle Count
Usage Cycles

Charge/discharge cycle count

Battery ID
Unique Identifier

Unique battery record ID

Supported Battery Chemistries

The simulator is configurable with four industry-standard lithium-ion battery chemistries.

LFP

Lithium Iron Phosphate

- High safety
- Long cycle life
- Lower energy density

Use: EVs, energy storage

NMC811

Nickel Manganese Cobalt

- High energy density
- Balanced performance
- Popular in passenger EVs

Use: High-range EVs

NCA

Nickel Cobalt Aluminum

- Very high energy density
- Excellent power output
- Moderate cycle life

Use: Performance EVs

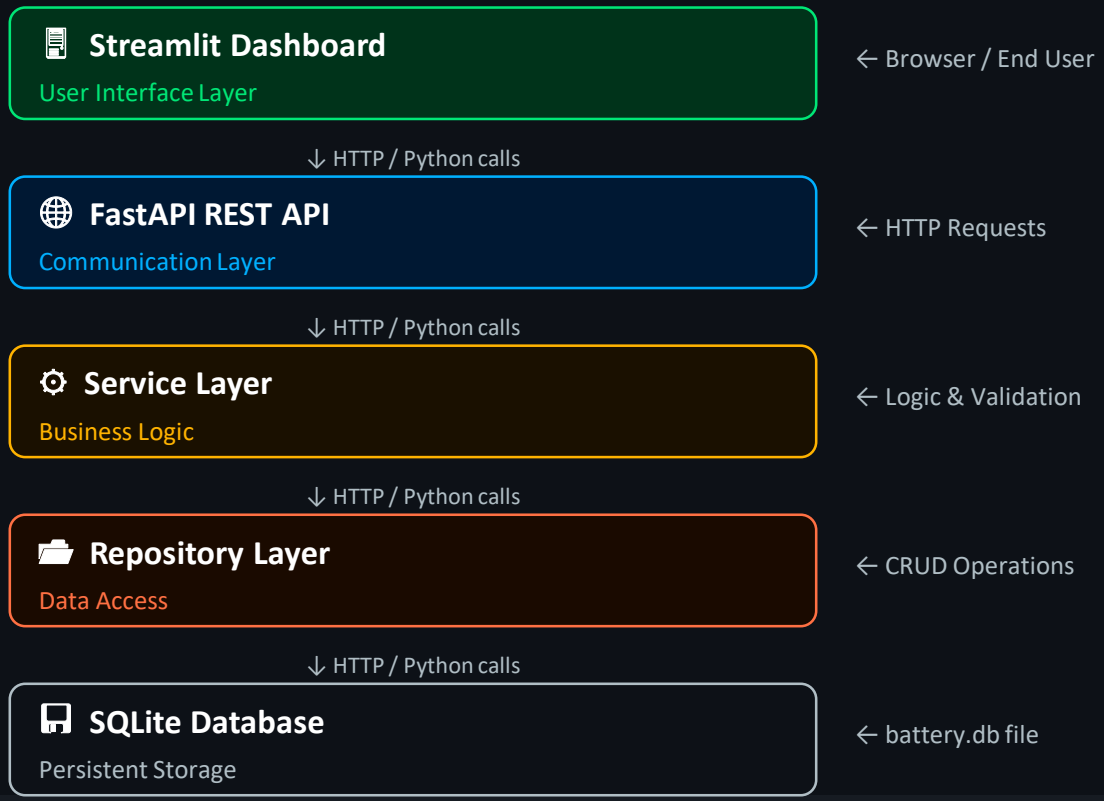
LCO

Lithium Cobalt Oxide

- High voltage
- High specific energy
- Lower thermal stability

Use: Consumer electronics

System Architecture



Code Structure & Organization

```
BatteryPack/
    ├── battery_pack/
    │   ├── generator/      ← Battery data generation logic
    │   ├── models/         ← Data models & schemas
    │   ├── services/       ← Business logic
    │   ├── repository/     ← Database operations
    │   └── simulation/     ← Core simulation engine
    ├── api/
    │   ├── routers/        ← API route definitions
    │   └── main.py         ← FastAPI entry point
    ├── database/
    │   ├── models.py       ← SQLAlchemy table models
    │   └── config.py       ← DB connection config
    ├── streamlit/
    │   └── dashboard.py    ← Streamlit UI
    └── battery.db          ← SQLite database file
```

Why Modular Architecture?

- Easier to debug — each layer has one job
- Reusable components across features
- Each module can be tested independently
- Multiple developers can work in parallel
- Scalable — add features without breaking others

Battery Data Generation

1. **User requests N batteries**
2. Battery Generator activated
3. Generate unique Battery IDs
4. Assign chemistry type
5. Simulate battery parameters
6. Create battery objects
7. **Return battery data**

Generated Parameters

`battery_id`	Unique battery identifier (UUID-based)
`chemistry`	LFP / NMC811 / NCA / LCO
`soc`	State of Charge: 0.0 – 1.0
`soh`	State of Health: 0.5 – 1.0
`voltage`	Cell voltage in Volts
`current`	Charge/discharge current in Amps
`temperature`	Cell temperature in °C
`capacity`	Energy capacity in Ah
`cycle_count`	Number of completed cycles

FastAPI Backend

FastAPI provides a high-performance REST API layer that connects the dashboard to application logic.

API Endpoints

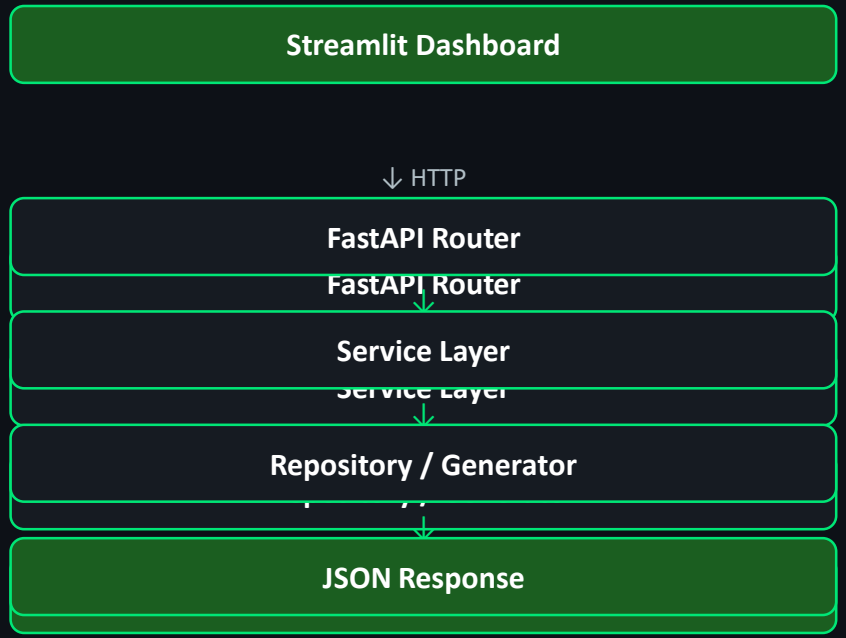
GET `/generate/{count}`
Generate N simulated batteries and store them

GET `/batteries/`
Retrieve all battery records from the database

GET `/battery/{battery_id}`
Fetch a specific battery by its unique ID

GET `/health`
Check API server health status

Request Lifecycle



Database Design — SQLite

Battery data is persisted in a SQLite database. The schema is simple, flat, and purpose-built for battery records.

TABLE: battery

Column	Type	Notes
id	INTEGER	PRIMARY KEY – auto-increment
battery_id	VARCHAR	UNIQUE – human-readable ID
chemistry	VARCHAR	LFP / NMC811 / NCA / LCO
soc	FLOAT	State of Charge (0.0 – 1.0)
soh	FLOAT	State of Health (0.5 – 1.0)
voltage	FLOAT	Cell voltage in Volts
current	FLOAT	Current in Amps
temperature	FLOAT	Temperature in °C
capacity	FLOAT	Capacity in Ah
cycle_count	INTEGER	Total charge/discharge cycles

Design Decisions

Primary Key
Auto-incremented integer id ensures each row is unique

Unique Battery ID
battery_id is a UNIQUE string for application-level queries

Float columns
SOC, SOH, voltage etc. stored as FLOAT for precision

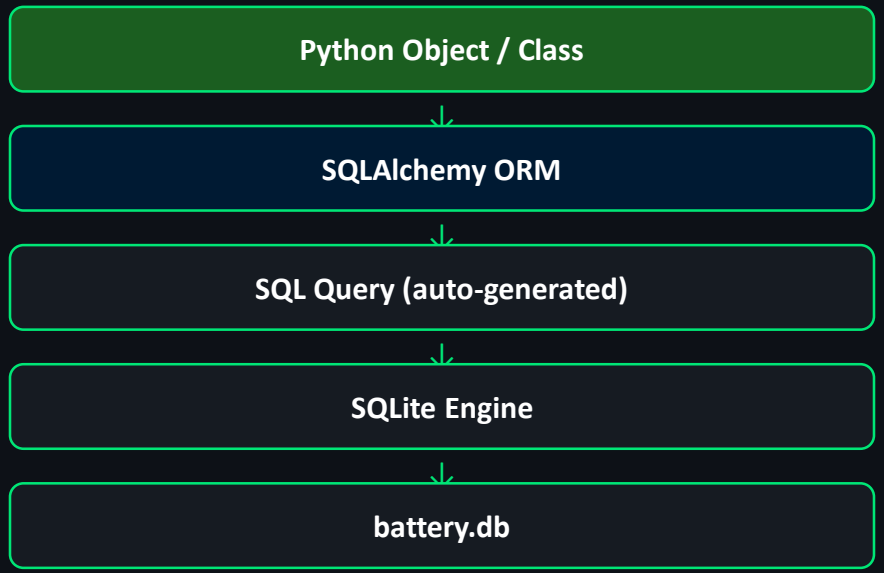
VARCHAR chemistry
Stores chemistry type as readable string

File-based DB
SQLite writes to battery.db — zero infrastructure needed

SQLAlchemy ORM

What is an ORM?

An Object-Relational Mapper lets you interact with a database using Python objects instead of writing raw SQL queries.



Why SQLAlchemy?

Pythonic
Define DB tables as Python classes

SQL Injection Safe
Parameterized queries by default

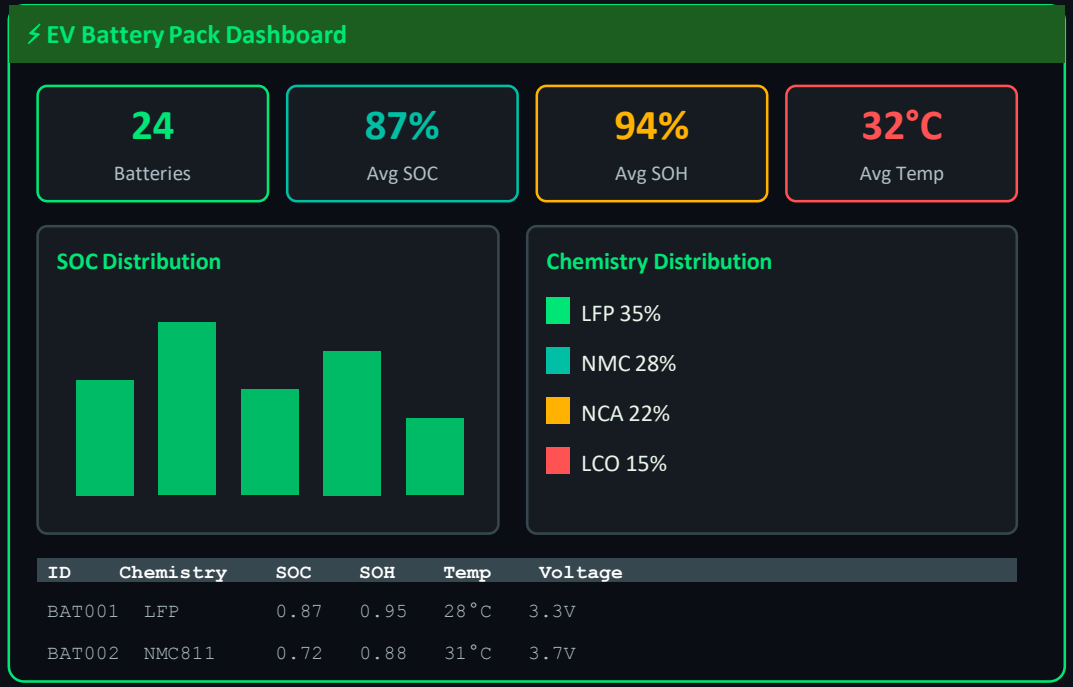
DB Portability
Switch from SQLite to PostgreSQL with minimal change

Session Management
Handles connections and transactions cleanly

Readable Queries
Filter, order, and query using Python methods

Streamlit Dashboard

The Streamlit dashboard provides the interactive user interface for the entire Battery Pack Simulator.



Dashboard Features

- ⚡ Generate battery batches
- View all battery records
- SOC distribution chart
- SOH distribution chart
- Temperature comparison
- Voltage comparison
- Chemistry breakdown
- Cycle count analysis
- Search by Battery ID

Data Visualization Pipeline

SQLite DB → battery.db

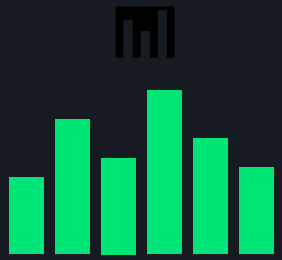
FastAPI → REST Response

Streamlit → HTTP Client

Pandas → DataFrame

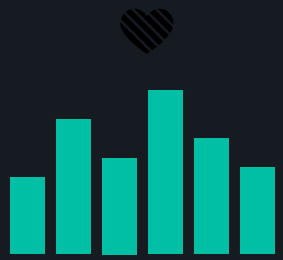
Plotly → Interactive Charts

Visualization Examples



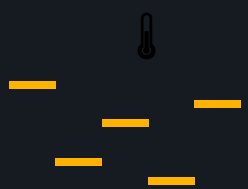
SOC Distribution

Bar Chart



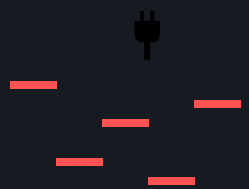
SOH Distribution

Histogram



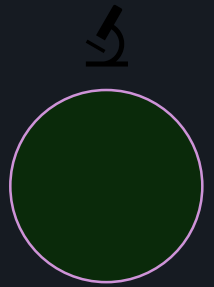
Temperature Range

Box Plot



Voltage by Chemistry

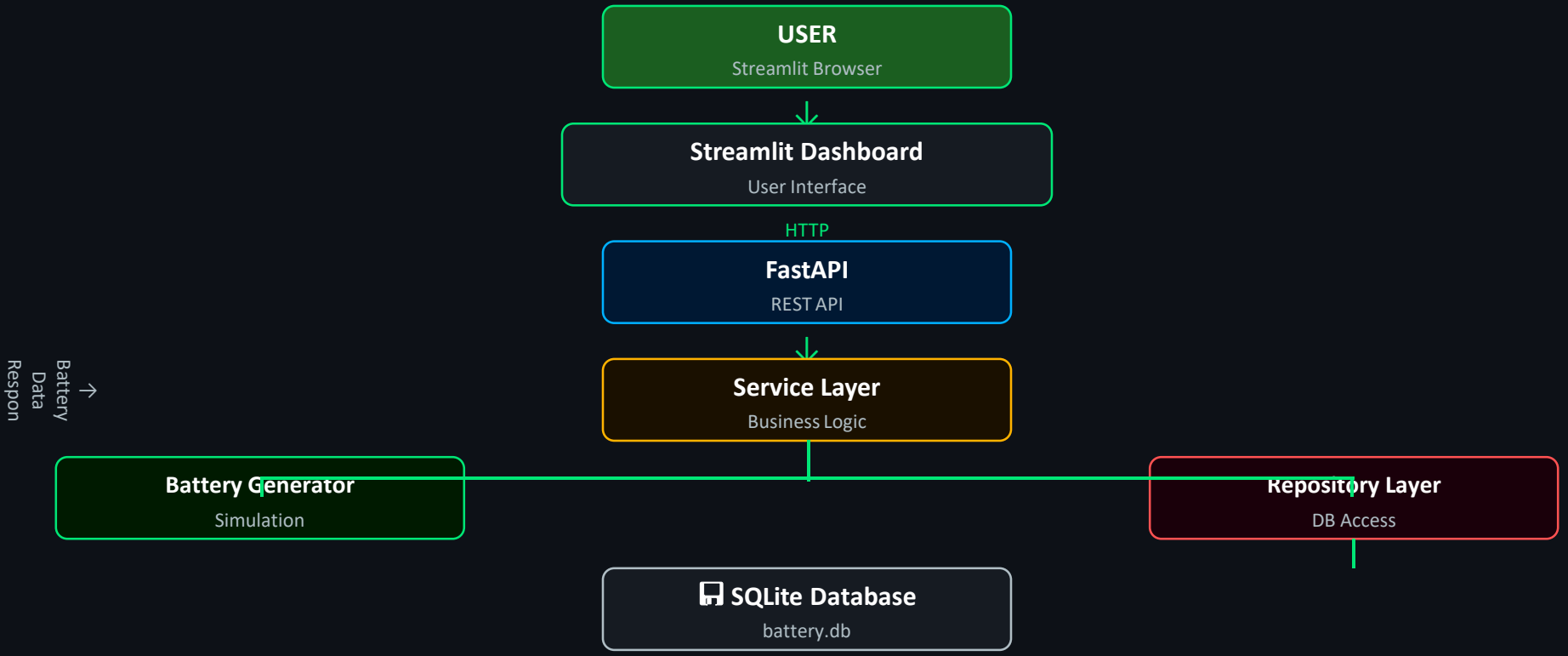
Grouped Bar



Chemistry Split

Pie / Donut

End-to-End Data Flow



⚡ EV SOCIETY

My Contributions

Backend Development

- Developed FastAPI route handlers
- Connected endpoints to service layer
- Implemented battery retrieval endpoints

Battery Data

- Worked with battery generation logic
- Handled all battery parameters
- Supported chemistry-specific simulation

Database Layer

- Designed battery database schema
- Integrated SQLite with SQLAlchemy
- Implemented repository-based persistence

Dashboard & UI

- Built complete Streamlit dashboard
- Connected dashboard to FastAPI over HTTP
- Added tables and Plotly visualizations

Architecture

- Separated all application layers
- Organized modules by responsibility
- Maintained clean project structure

Challenges & Solutions

CHALLENGE	SOLUTION
⚠ Understanding the battery simulation architecture	✓ Studied the existing codebase systematically and separated responsibilities into manageable modules
⚠ Connecting Streamlit with FastAPI	✓ Used Python requests library to make HTTP calls from Streamlit to FastAPI endpoints
⚠ Persisting battery data reliably	✓ Designed a clean SQLite schema and integrated SQLAlchemy for structured ORM-based access
⚠ Displaying raw data meaningfully	✓ Used Pandas DataFrames to structure data and Plotly for interactive, interpretable charts
⚠ Maintaining clean project structure	✓ Separated API, services, repository, generator, database, and UI into distinct modules

Learning Outcomes

Technical Skills

- Python
- FastAPI
- REST APIs
- SQLite
- SQLAlchemy
- Streamlit
- Pandas
- Plotly
- Git / GitHub

Software Engineering

- Layered architecture
- Modular programming
- API design patterns
- Database design
- Frontend-backend integration
- Debugging techniques
- Project organization

Domain Knowledge

- Battery SOC & SOH
- Battery chemistries
- Thermal behavior
- Charge / discharge cycles
- Electrical parameters
- EV battery systems

Future Scope & Conclusion

Future Improvements

- ⚡ Real-time battery simulation
- Charging / discharging cycles
- Advanced SOC / SOH models
- Drive-cycle simulation
- Battery failure detection
- Advanced thermal modeling
- PostgreSQL support
- Dockerization & deployment
- User authentication
- Cloud deployment
- CSV / Excel export

Conclusion

This project transformed a battery simulation engine into a complete full-stack software application.

- **Python**
- **FastAPI**
- **SQLite + SQLAlchemy**
- **Streamlit**
- **Plotly + Pandas**

Practical experience in both battery simulation concepts and full-stack software development.

⚡ EV SOCIETY ⚡

Thank You

Questions?

Python | FastAPI | SQLite | SQLAlchemy | Streamlit | Plotly

⚡ 71% Battery Charged

Prateek | Internship Project | EV Battery Pack Simulator